Dobot M1 API

接口说明

文档版本：V1.0

发布日期：2018-03-23

深圳市越疆科技有限公司

免责申明

在法律允许的最大范围内，本手册所描述的产品（含其硬件、软件、固件等）均“按照现状”提供，可能存在瑕疵、错误或故障，越疆不提供任何形式的明示或默示保证，包括但不限于适销性、质量满意度、适合特定目的、不侵犯第三方权利等保证；亦不对使用本手册或使用本公司产品导致的任何特殊、附带、偶然或间接的损害进行赔偿。

在使用本产品前详细阅读本使用手册及网上发布的相关技术文档并了解相关信息，确保在充分了解机器人及其相关知识的前提下使用机械臂。越疆建议您在专业人员的指导下使用本手册。该手册所包含的所有安全方面的信息都不得视为Dobot的保证，即便遵循本手册及相关说明，使用过程中造成的危害或损失依然有可能发生。

本产品的使用者有责任确保遵循相关国家的切实可行的法律法规，确保在越疆机械臂的使用中不存在任何重大危险。

越疆科技有限公司

地址：深圳市南山区桃源街道塘朗工业区A区8栋4楼

网址：http://cn.dobot.cc/

前 言

目的

本文档旨在对 Dobot API 接口进行详细说明，并给出基于 Dobot API 接口开发应用程序的一般流程。

读者对象

本手册适用于：

- 客户工程师
- 安装调测工程师
- 技术支持工程师

修订记录

时间	修订记录
2018/03/23	第一次发布

符号约定

在本手册中可能出现下列标志，它们所代表的含义如下。

符号	说明
⚠危险	表示有高度潜在危险，如果不能避免，会导致人员死亡或严重伤害
⚠警告	表示有中度或低度潜在危害，如果不能避免，可能导致人员轻微伤害、机械臂毁坏等情况
⚠注意	表示有潜在风险，如果忽视这些文本，可能导致机械臂损坏、数据丢失或不可预知的结果
📖说明	表示是正文的附加信息，是对正文的强调和补充

目 录

1. API 接口说明

1.1 Dobot 指令简介

控制器支持两种类型的指令：立即指令与队列指令。

- 立即指令：Dobot控制器在收到指令后立即处理该指令，而不管当前控制器是否在还在处理其他指令。
- 队列指令：Dobot控制器在收到指令后会将该指令放入控制器内部的指令队列中，Dobot控制器将顺序执行指令。

关于Dobot指令更具体的内容，请参见《Dobot通信协议》。

1.2 指令超时

1.2.1 设置指令超时时间

如*1.1 Dobot指令简介*中介绍，发送给Dobot控制器的所有指令都带有返回。当通信链路干扰等问题造成指令错误时，控制器将无法识别该条指令且无法返回。因此，每条下发给控制器的指令都可设置一个超时时间。该指令超时时间可以通过以下的API进行设置。

表 1.1 设置指令超时时间

原型	int SetCmdTimeout(uint32_t cmdTimeout)
功能	设置指令超时时间。当下发一条指令后如果需在规定时间内返回，则可调用此接口设置超时时间，判断该指令是否超时
功能	cmdTimeout：指令超时时间。单位：ms
返回	DobotCommunicate_NoError：无错误

1.3 连接/断开

1.3.1 搜索 Dobot

表 1.2 搜索 Dobot

原型	int SearchDobot(char *dobotList, uint32_t maxLen)
功能	搜索Dobot，动态库将搜索到的Dobot信息存储，并使用ConnectDobot连接Dobot
功能	dobotList：字符串数组，Dobot动态库会将搜索到的串口或UDP信息写入到dobotList。一个典型的dobotList的格式为"COM1 COM3 COM6 192.168.0.5"，不同的串口或IP地址以空格分开 maxLen：字符串最大长度，以避免内存溢出
返回	Dobot 数量

1.3.2 连接 Dobot

表 1.3 连接 Dobot 控制器接口说明

原型	int ConnectDobot(const char *portName, uint32_t baudrate, char *fwType, char *version)
功能	连接 Dobot。其中，portName 可以从 int SearchDobot(char *dobotList, uint32_t maxLen)的 char *dobotList 中得到 若 portName 为空，并直接调用 ConnectDobot，则动态库将自动连接随机搜索到的 Dobot
参数	portName：Dobot 端口名。对于串口，其值为“COM3”。对于 UDP，可能是“192.168.0.5” baudrate：波特率 fwType：固件类型，包括：Dobot 和 Marlin version：版本号
返回	DobotConnect_NoError：连接 Dobot 成功 DobotConnect_NotFound：未找到 Dobot 端口 DobotConnect_Occupied：Dobot 端口被占用

⚠注意

请提前安装所需的驱动以便使API接口能识别Dobot控制器接口，详情请查询Dobot用户手册。

1.3.3 断开 Dobot

表 1.4 断开 Dobot 接口说明

原型	int DisconnectDobot(void)
功能	断开 Dobot
参数	无
返回	DobotConnect_NoError：无错误

1.3.4 示例：连接示例

程序 1.1 连接示例

```
#include "DobotDll.h"


int split(char **dst, char* str, const char* spl)
{
    int n = 0;
    char *result = NULL;
```

```
    result = strtok(str, spl);
    while( result != NULL )
    {
        strcpy(dst[n++], result);
        result = strtok(NULL, spl);
    }
    return n;
}

int main(void)
{
    int maxDevCount = 100;
    int maxDevLen = 20;

    char *devsChr = new char[maxDevCount * maxDevLen]();
    char **devsList = new char*[maxDevCount]();
    for(int i=0; i<maxDevCount; i++)
        devsList[i] = new char[maxDevLen]();

    SearchDobot(devsChr, 1024);
    split(devsList, devsChr, " ");
    ConnectDobot(devsList[0], 115200, NULL, NULL, NULL);

//控制 Dobot

    DisconnectDobot();

    delete[] devsChr;
    for(int i=0; i<maxDevCount; i++)
        delete[] devsList[i];
    delete[] devsList;
}
```

1.4 指令队列控制

Dobot控制器中有一个存放指令的队列，以达到顺序储存和执行指令的目的。同时，通过启动和停止指令，实现丰富的异步操作。

⚠注意

只有将“isQueued”参数设置为“1”的指令才能加入指令队列。

1.4.1 执行队列中的指令

表 1.5 执行指令

原型	int SetQueuedCmdStartExec(void)
功能	Dobot控制器开始循环查询指令队列，如果队列中有指令，则顺序取出并执行，执行完一条指令后才会取出下一条继续执行
参数	无
返回	DobotCommunicate_NoError：指令正常返回 DobotCommunicate_Timeout：指令无返回，导致超时

1.4.2 停止执行队列中的指令

表 1.6 停止执行指令

原型	int SetQueuedCmdStopExec(void)
功能	Dobot 控制器停止循环查询队列并停止执行指令。在停止过程中若 Dobot 控制器正在执行一条指令，则待该指令执行完成后再停止。
参数	无
返回	DobotCommunicate_NoError：指令正常返回 DobotCommunicate_Timeout：指令无返回，导致超时

1.4.3 强制停止执行队列中的指令

表 1.7 强制停止执行指令

原型	int SetQueuedCmdForceStopExec(void)
功能	Dobot 控制器停止循环查询队列并停止执行指令。在停止过程中若 Dobot 控制器正在执行一条指令，则该指令将会停止执行。
参数	无
返回	DobotCommunicate_NoError：指令正常返回 DobotCommunicate_Timeout：指令无返回，导致超时

1.4.4 示例：同步处理 PTP 指令下发和队列控制

PTP详细说明请参见*1.11 PTP功能*。

程序 1.2 同步处理 PTP 指令下发和队列控制

```
#include "DobotDll.h"


int main(void)
{
    uint64_t queuedCmdIndex = 0;
    PTPCmd      cmd;

    cmd.ptpMode = 0;
    cmd.x           = 200;
    cmd.y           = 0;
    cmd.z           = 0;
    cmd.r           = 0;

    ConnectDobot(NULL, 115200, NULL, NULL, NULL);

    SetQueuedCmdStartExec();
    SetPTPCmd(&cmd, true, &queuedCmdIndex);
    SetQueuedCmdStopExec();

    DisconnectDobot();
}
```

1.4.5 示例：异步处理 PTP 指令下发和队列控制

程序 1.3 异步处理 PTP 指令下发和队列控制

```
#include "DobotDll.h"


// 主线程
int main(void)
{
    ConnectDobot(NULL, 115200, NULL, NULL, NULL);
}


int onButtonClick()
{
    static bool flag = True;
```

```
    if (flag)
        SetQueuedCmdStartExec();
    else
        SetQueuedCmdStopExec();
}

// 子线程
int thread(void)
{
    uint64_t queuedCmdIndex = 0;
    PTPCmd    cmd;

    cmd.ptpMode = 0;
    cmd.x       = 200;
    cmd.y       = 0;
    cmd.z       = 0;
    cmd.r       = 0;

    while(true)
        SetPTPCmd(&cmd, true, &queuedCmdIndex);
}
```

1.4.6 清空指令队列

该接口可以清空Dobot控制器中的指令队列。

表 1.8 清除指令队列

原型	int SetQueuedCmdClear(void)
功能	清空指令队列
参数	无
返回	DobotCommunicate_NoError：指令正常返回 DobotCommunicate_Timeout：指令无返回，导致超时

1.4.7 获取指令索引

在 Dobot 控制器指令队列机制中，有一个 64 位内部计数器。当控制器每执行完一条指令时，该计数器将自动加一。通过该指令，可以查询当前执行完成的指令的索引。

表 1.9 获取指令队列当前索引接口说明

原型	int GetQueuedCmdCurrentIndex(uint64_t *queuedCmdCurrentIndex)
功能	获取当前执行完成的指令的索引
参数	queuedCmdCurrentIndex：指令索引
返回	DobotCommunicate_NoError：指令正常返回 DobotCommunicate_Timeout：指令无返回，导致超时

1.4.8 示例：获取指令索引实现运动同步

程序 1.4 获取指令索引实现运动同步

```
#include "DobotDll.h"


int main(void)
{
    uint64_t queuedCmdIndex     = 0;
    uint64_t executedCmdIndex = 0;
    PTPCmd     cmd;


    cmd.ptpMode = 0;
    cmd.x            = 200;
    cmd.y            = 0;
    cmd.z            = 0;
    cmd.r            = 0;


    ConnectDobot(NULL, 115200, NULL, NULL, NULL);


    SetQueuedCmdStartExec();
    SetPTPCmd(&cmd, true, &queuedCmdIndex);


// 通过比较队列索引实现运动同步
While(executedCmdIndex
 < queuedCmdIndex)            GetQueuedCmdCurrentIndex(&executedCmdIndex);
    SetQueuedCmdStopExec();
    DisconnectDobot();
}
```

1.5 设备信息

1.5.1 设置设备序列号

表 1.10 设置设备序列号

原型	int SetDeviceSN(const char *deviceSN)
功能	设置设备序列号。该接口仅在出厂时有效（需要特殊密码）
参数	deviceSN：字符串指针
返回	DobotCommunicate_NoError：指令正常返回 DobotCommunicate_Timeout：指令无返回，导致超时

1.5.2 获取设备序列号

表 1.11 获取设备序列号

原型	int GetDeviceSN(char *deviceSN, uint32_t maxLen)
功能	获取设备序列号
参数	deviceSN：字符串指针 maxLen：字符串最大长度，以避免溢出`
返回	DobotCommunicate_NoError：指令正常返回 DobotCommunicate_Timeout：指令无返回，导致超时

1.5.3 设置设备名称

表 1.12 设置设备名称

原型	int SetDeviceName(const char *deviceName)
功能	设置设备名称。当有多台机器时，可调用该接口设置设备名以作区分
参数	deviceName：字符串指针
返回	DobotCommunicate_NoError：指令正常返回 DobotCommunicate_Timeout：指令无返回，导致超时

1.5.4 获取设备名称

表 1.13 获取设备名称

原型	int GetDeviceName(char *deviceName, uint32_t maxLen)
功能	获取设备名称。当有多台机器时，可调用该接口设置设备名以作区分
参数	deviceName：字符串指针

	maxLen：字符串最大长度，以避免溢出
返回	DobotCommunicate_NoError：指令正常返回 DobotCommunicate_Timeout：指令无返回，导致超时

1.5.5 获取设备版本信息

表 1.14 获取设备版本信息

原型	int GetDeviceVersion(uint8_t typeIndex, uint8_t *majorVersion, uint8_t *minorVersion, uint8_t *revision)
描述	获取设备版本信息

参数	typeIndex：版本类型
	enum FirmwareType{
	NO_SWITCH,
	DOBOT_SWITCH,
	PRINTING_SWITCH,
	DRIVER1_SWITCH,
	DRIVER2_SWITCH,
	DRIVER3_SWITCH,
	DRIVER4_SWITCH,
	DRIVER5_SWITCH,
	FPGA_SWITCH,
	SWITCH_FM_MAX
	};
	majorVersion：主版本
	minorVersion：次版本
	revision：修订版本

返回	DobotCommunicate_NoError：指令正常返回 DobotCommunicate_Timeout：指令无返回，导致超时

1.6 实时位姿

在Dobot M1中，Dobot控制器根据各关节轴的编码器读数计算实时位姿的基准值。在控制Dobot时，Dobot控制器将基于实时位姿的基准值，以及实时运动状态，更新实时位姿。

1.6.1 获取机械臂实时位姿

表 1.15 获取实时位姿

原型	int GetPose(Pose *pose)
功能	获取机械臂实时位姿

参数	Pose 定义：
	typedef struct tagPose {
	float x; //机械臂坐标系 x
	float y; //机械臂坐标系 y
	float z; //机械臂坐标系 z
	float r; //机械臂坐标系 r
	float jointAngle[4]; //机械臂关节轴(J1、J2、J3、J4)角度
	}Pose;
	pose：Pose 指针

返回	DobotCommunicate_NoError：指令正常返回 DobotCommunicate_Timeout：指令无返回，导致超时

1.7 报警功能

1.7.1 获取系统报警状态

表 1.16 获取系统报警状态

原型	int GetAlarmsState(uint8_t *alarmsState, uint32_t *len, unsigned int maxLen)
功能	获取系统报警状态
参数	alarmsState：数组首地址。每一个字节可以标识8个报警项的报警状态，且MSB（Most Significant Bit）在高位，LSB（Least Significant Bit）在低位 len：报警所占字节 maxLen：数组最大长度，以避免溢出
返回	DobotCommunicate_NoError：指令正常返回 DobotCommunicate_Timeout：指令无返回，导致超时

1.7.2 清除系统所有报警

表 1.17 清除系统所有报警

原型	int ClearAllAlarmsState(void)
功能	清除系统所有报警
参数	无
返回	DobotCommunicate_NoError：指令正常返回 DobotCommunicate_Timeout：指令无返回，导致超时

1.8 回零功能

如果机械臂运行速度过快或者负载过大可能会导致精度降低，此时可执行回零操作，提高精度。

1.8.1 设置回零位置

表 1.18 设置回零参数

原型	int SetHOMEParams(HOMEParams *homeParams, bool isQueued, uint64_t *queuedCmdIndex)
功能	设置回零位置

参数	HOMEParams 定义：
	typedef struct tagHOMEParams {
	float x; //机械臂坐标系 x
	float y; //机械臂坐标系 y
	float z; //机械臂坐标系 z
	float r; //机械臂坐标系 r
	}HOMEParams;
	homeParams：HOMEParams 指针
	isQueued：是否将该指令加入指令队列
	queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义

返回	DobotCommunicate_NoError：指令正常返回 DobotCommunicate_BufferFull：指令队列已满 DobotCommunicate_Timeout：指令无返回，导致超时

1.8.2 获取回零位置

表 1.19 获取回零参数

原型	int GetHOMEParams(HOMEParams *homeParams)
功能	获取回零位置

参数	HOMEParams 定义：
	typedef struct tagHOMEParams {
	float x; //机械臂坐标系 x
	float y; //机械臂坐标系 y
	float z; //机械臂坐标系 z
	float r; //机械臂坐标系 r
	}HOMEParams;
	homeParams：HOMEParams 指针

返回	DobotCommunicate_NoError：指令正常返回

	DobotCommunicate_Timeout：指令无返回，导致超时

1.8.3 执行回零功能

表 1.20 执行回零功能

原型	int SetHOMECmd(HOMECmd *homeCmd, bool isQueued, uint64_t *queuedCmdIndex)
功能	执行回零功能。调用该接口前如果未调用 SetHOMEParams 接口，则表示直接回零至系统设置的位置。如果调用了 SetHOMEParams 接口，则回零至用户自定义的位置

参数	HOMECmd 定义：
	typedef struct tagHOMECmd {
	uint32_t reserved; //保留
	}HOMECmd;
	homeCmd：HOMECmd 指针
	isQueued：是否将该指令加入指令队列
	queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义

返回	DobotCommunicate_NoError：指令正常返回 DobotCommunicate_BufferFull：指令队列已满 DobotCommunicate_Timeout：指令无返回，导致超时

1.9 ARM 方向

1.9.1 设置机械臂方向

表 1.21 设置机械臂方向

原型	int SetArmOrientation(ArmOrientation armOrientation, bool isQueued, uint64_t *queuedCmdIndex)
描述	设置机械臂方向
参数	ArmOrientation定义： typedef enum tagArmOrientation { LeftyArmOrientation, //左手方向 RightyArmOrientation //右手方向 }ArmOrientation; armOrientation：ArmOrientation枚举 isQueued：是否将该指令加入指令队列 queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否

	则，该参数无意义
返回	DobotCommunicate_NoError：指令正常返回 DobotCommunicate_BufferFull：指令队列已满 DobotCommunicate_Timeout：指令无返回，导致超时

1.9.2 获取机械臂方向

表 1.22 获取机械臂方向

原型	int GetArmOrientation(ArmOrientation *armOrientation)
描述	获取机械臂方向
参数	ArmOrientation定义： typedef enum tagArmOrientation { LeftyArmOrientation, //左手方向 RightyArmOrientation //右手方向 }ArmOrientation; armOrientation：ArmOrientation枚举
返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_Timeout:指令无返回，导致超时

1.10 JOG 功能

1.10.1 设置点动速度百分比和加速度百分比

表 1.23 设置点动速度百分比和加速度百分比

原型	int SetJOGCommonParams(JOGCommonParams *jogCommonParams, bool isQueued, uint64_t *queuedCmdIndex)
功能	设置点动（关节坐标系下和笛卡尔坐标系下）速度百分比和加速度百分比

参数	JOGCommonParams 定义：
	typedef struct tagJOGCommonParams {
	float velocityRatio; //速度比例，关节坐标轴点动和笛卡尔坐标轴点动共用
	float accelerationRatio; //加速度比例，关节坐标轴点动和笛卡尔坐标轴点动共用
	}JOGCommonParams;
	jogCommonParams：JOGCommonParams 指针
	isQueued：是否将该指令加入指令队列
	queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。

	否则，该参数无意义
返回	DobotCommunicate_NoError：指令正常返回 DobotCommunicate_BufferFull：指令队列已满 DobotCommunicate_Timeout：指令无返回，导致超时

1.10.2 获取点动速度百分比和加速度百分比

表 1.24 获取点动速度百分比和加速度百分比

原型	int GetJOGCommonParams(JOGCommonParams *jogCommonParams)
功能	获取点动（关节坐标系下和笛卡尔坐标系下）速度百分比和加速度百分比

参数	JOGCommonParams 定义：
	typedef struct tagJOGCommonParams {
	float velocityRatio; //速度比例，关节点动和坐标轴点动共用
	float accelerationRatio; //加速度比例，关节点动和坐标轴点动共用
	}JOGCommonParams;
	jogCommonParams：JOGCommonParams 指针

返回	DobotCommunicate_NoError：指令正常返回 DobotCommunicate_Timeout：指令无返回，导致超时

1.10.3 执行点动指令

表 1.25 执行点动指令接口说明

原型	int SetJOGCmd(JOGCmd *jogCmd, bool isQueued, uint64_t *queuedCmdIndex)
功能	执行点动指令。设置点动相关参数后可调用该接口

参数	JOGCmd 定义：
	typedef struct tagJOGCmd {
	uint8_t isJoint; //点动方式：0，笛卡尔坐标轴点动；1，关节点动
	uint8_t cmd; //点动命令（取值范围 0~10）
	}JOGCmd;
	//点动命令详细说明
	enum {
	IDLE, //空闲状态
	AP_DOWN, //X+/Joint1+
	AN_DOWN, //X-/Joint1-
	BP_DOWN, //Y+/Joint2+
	BN_DOWN, //Y-/Joint2-
	CP_DOWN, //Z+/Joint3+

	CN_DOWN, //Z-/Joint3- DP_DOWN, //R+/Joint4+ DN_DOWN, //R-/Joint4- LP_DOWN, //L+ LN_DOWN //L- }; jogCmd：JOGCmd 指针 isQueued：是否将该指令加入指令队列 queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义
返回	DobotCommunicate_NoError：指令正常返回 DobotCommunicate_BufferFull：指令队列已满 DobotCommunicate_Timeout：指令无返回，导致超时

1.11 PTP 功能

PTP点位模式点位模式即实现点到点运动，Dobot M1的点位模式包括MOVJ、MOVL以及JUMP三种运动模式。不同的运动模式，示教后存点回放的运动轨迹不同。

- MOVJ：关节运动，由A点运动到B点，各个关节从A点对应的关节角运行至B点对应的关节角。关节运动过程中，各个关节轴的运行时间需一致，且同时到达终点，如图 1.1所示。

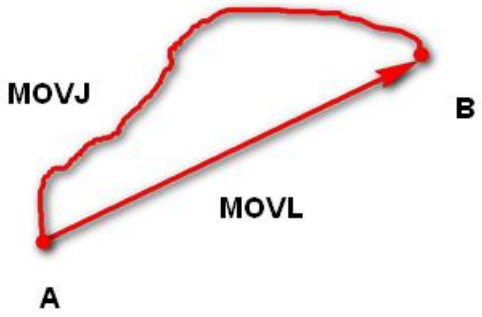


图 1.1 MOVL 和 MOVJ 运动模式

- MOVL：直线运动，A点到B点的路径为直线，如图 1.1所示。
- JUMP：门型轨迹，A点到B点以MOVJ运动模式移动，如图 1.2所示。
    1. 以MOVJ运动模式上升到一定高度（Height）。
    2. 以MOVJ运动模式过渡到最大抬升高度（Limit）。
    3. 以MOVJ运动模式平移到B点上方的高度处。
    4. 以MOVJ运动模式过渡到B点高度加上Height后的高度处。
    5. 以MOVJ运动模式下降到B点所在位置。

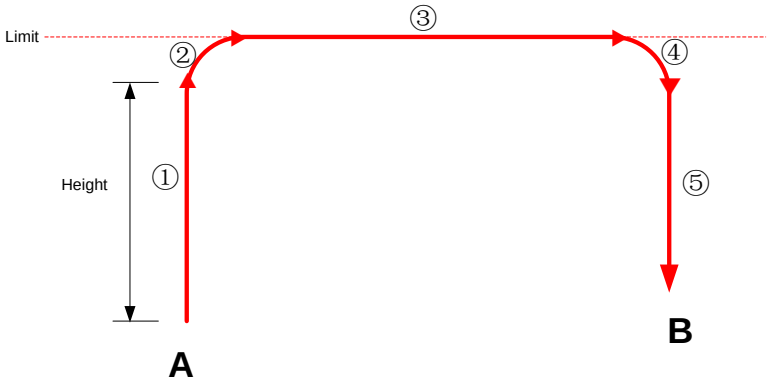
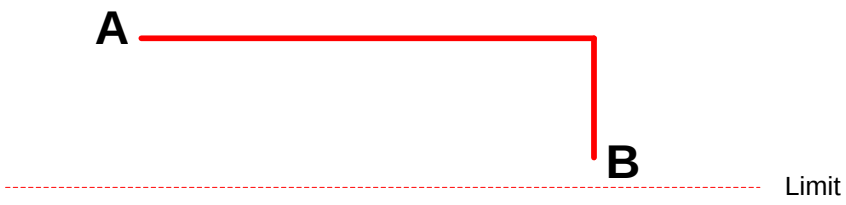


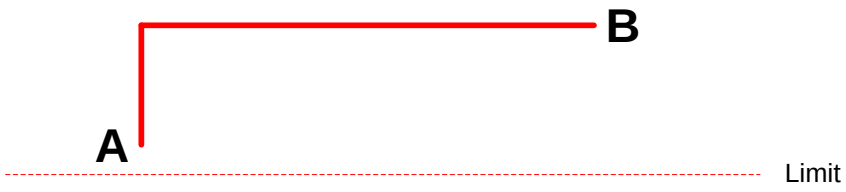
图 1.2 JUMP 运动模式

JUMP运动模式下，如果起始点或结束点高度大于等于最大抬升高度，或者起始点抬升到一定高度后，大于等于最大抬升高度，其运动轨迹与图 1.2有所不同。假设A为起始点，B为结束点，Limit为最大抬升高度，Height为抬升高度。

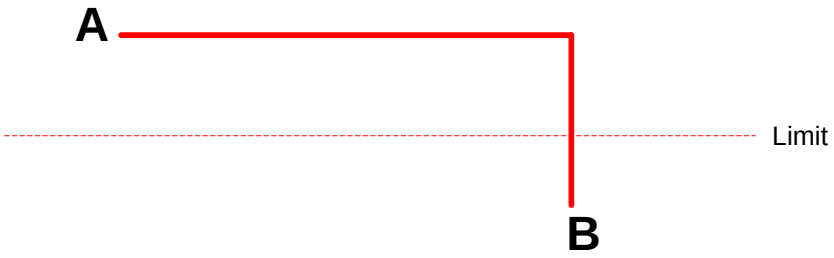
- A点、B点高度均大于Limit，且A点高度大于B点高度。



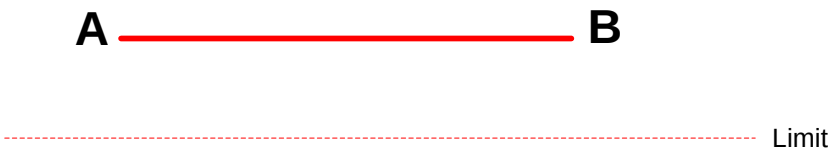
- A点、B点高度均大于Limit，且A点高度小于B点高度。



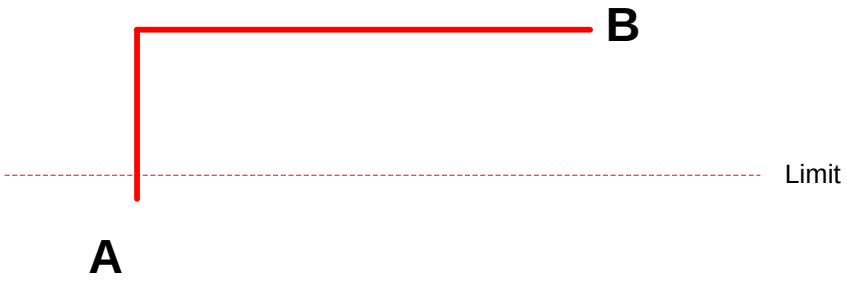
- A点高度大于Limit，B点高度小于Limit。



- A点、B点高度相同，且大于Limit。



- A点高度小于Limit，B点高度大于Limit。



- A点、B点高度与Limit相同。

- A点、B点高度均小于Limit，A点高度+Height和B点+Height大于Limit。

1.11.1 设置 JUMP 模式下抬升高度和最大抬升高度

表 1.26 设置 JUMP 模式下抬升高度和最大抬升高度

原型	int SetPTPJumpParams(PTPJumpParams *ptpJumpParams, bool isQueued, uint64_t *queuedCmdIndex)
功能	设置 JUMP 模式下抬升高度和最大抬升高度

参数	PTPJumpParams 定义：
	typedef struct tagPTPJumpParams {
	float jumpHeight; //抬升高度
	float zLimit; //最大抬升高度
	}PTPJumpParams;
	ptpJumpParams：PTPJumpParams 指针
	isQueued：是否将该指令指定为队列命令
	queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义

返回	DobotCommunicate_NoError：指令正常返回 DobotCommunicate_BufferFull：指令队列已满 DobotCommunicate_Timeout：指令无返回，导致超时

1.11.2 获取 JUMP 模式下抬升高度和最大抬升高度

表 1.27 JUMP 模式下抬升高度和最大抬升高度

原型	int GetPTPJumpParams(PTPJumpParams *ptpJumpParams)
功能	获取 JUMP 模式下抬升高度和最大抬升高度

参数	PTPJumpParams 定义：
	typedef struct tagPTPJumpParams {
	float jumpHeight; //抬升高度
	float zLimit; //最大抬升高度

	}PTPJumpParams; ptpJumpParams：PTPJumpParams 指针
返回	DobotCommunicate_NoError：指令正常返回 DobotCommunicate_Timeout：指令无返回，导致超时

1.11.3 设置 PTP 运动的速度百分比和加速度百分比

表 1.28 设置 PTP 运动的速度百分比和加速度百分比

原型	int SetPTPCommonParams(PTPCommonParams *ptpCommonParams, bool isQueued, uint64_t *queuedCmdIndex)
功能	设置 PTP 运动的速度百分比和加速度百分比

参数	PTPCommonParams 定义：
	typedef struct tagPTPCommonParams {
	float velocityRatio; //PTP 模式速度百分比，关节坐标轴和笛卡尔坐标轴共用
	float accelerationRatio; //PTP 模式加速度百分比，关节坐标轴和笛卡尔坐标轴共用
	}PTPCommonParams;
	ptpCommonParams：PTPCommonParams 指针
	isQueued：是否将该指令指定为队列命令
	queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义

返回	DobotCommunicate_NoError：指令正常返回 DobotCommunicate_BufferFull：指令队列已满 DobotCommunicate_Timeout：指令无返回，导致超时

1.11.4 获取 PTP 运动的速度百分比和加速度百分比

表 1.29 PTP 运动的速度百分比和加速度百分比

原型	int GetPTPCommonParams(PTPCommonParams *ptpCommonParams)
功能	PTP 运动的速度百分比和加速度百分比

参数	PTPCommonParams 定义：
	typedef struct tagPTPCommonParams {
	float velocityRatio; //PTP 模式速度百分比，关节坐标轴和笛卡尔坐标轴共用
	float accelerationRatio; //PTP 模式加速度百分比，关节坐标轴和笛卡尔坐标轴共用

	}PTPCommonParams; ptpCommonParams：PTPCommonParams 指针
返回	DobotCommunicate_NoError：指令正常返回 DobotCommunicate_Timeout：指令无返回，导致超时

1.11.5 执行 PTP 指令

表 1.30 执行 PTP 指令

原型	int SetPTPCmd(PTPCmd *ptpCmd, bool isQueued, uint64_t *queuedCmdIndex)
功能	执行 PTP 指令。设置 PTP 相关参数后，调用此函数可使机械臂运动至设置的目标点。

参数	PTPCmd 定义：
	typedef struct tagPTPCmd {
	uint8_t ptpMode; //PTP 模式，取值范围：0~9
	float x; //（x,y,z,r）为坐标参数，可为笛卡尔坐标、关节坐标、笛卡尔坐标增量或关节坐标增量
	float y;
	float z;
	float r;
	}PTPCmd;
	//其中，ptpMode 取值如下：
	enum {
	JUMP_XYZ, //JUMP 模式，（x,y,z,r）为笛卡尔坐标系下的目标点坐标
	MOVJ_XYZ, //MOVJ 模式，（x,y,z,r）为笛卡尔坐标系下的目标点坐标
	MOVL_XYZ, //MOVL 模式，（x,y,z,r）为笛卡尔坐标系下的目标点坐标
	JUMP_ANGLE, //JUMP 模式，（x,y,z,r）为关节坐标系下的目标点坐标
	MOVJ_ANGLE, //MOVJ 模式，（x,y,z,r）为关节坐标系下的目标点坐标
	MOVL_ANGLE, //MOVL 模式，（x,y,z,r）为关节坐标系下的目标点坐标
	MOVJ_INC, //MOVJ 模式，（x,y,z,r）为关节坐标系下的坐标增量
	MOVL_INC, //MOVL 模式，（x,y,z,r）为笛卡尔坐标系下的坐标增量

	MOVJ_XYZ_INC, //MOVJ 模式，（x,y,z,r）为笛卡尔坐标系下的坐标增量 JUMP_MOVL_XYZ, //JUMP 模式，平移时运动模式为 MOVL。（x,y,z,r）为笛卡尔坐标系下的坐标增量 }; ptpCmd：PTPCmd 指针 isQueued：是否将该指令指定为队列命令 queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义
返回	DobotCommunicate_NoError：指令正常返回 DobotCommunicate_BufferFull：指令队列已满 DobotCommunicate_Timeout：指令无返回，导致超时

1.12 CP 功能

CP即连续运动轨迹。

1.12.1 执行 CP 指令

表 1.31 执行 CP 指令

原型	int SetCPCmd(CPCmd *cpCmd, bool isQueued, uint64_t *queuedCmdIndex)
功能	执行 CP 指令

参数	CPCmd 定义：
	typedef struct tagCPCmd {
	uint8_t cpMode; //CP 模式。0：相对模式，表示相对距离，即笛卡尔坐标增量。1：绝对模式，表示绝对距离，即笛卡尔坐标系下目标点坐标
	float x; //x,y,z 可以设置为坐标增量，也可设置为目的坐标点
	float y;
	float z;
	union {
	float velocity; //保留
	float power; //保留
	};
	}CPCmd;
	cpCmd：CPCmd 指针
	isQueued：是否将该指令指定为队列命令
	queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。

	否则，该参数无意义
返回	DobotCommunicate_NoError：指令正常返回 DobotCommunicate_BufferFull：指令队列已满 DobotCommunicate_Timeout：指令无返回，导致超时

⚠注意

当指令队列中有多条连续的CP指令时，Dobot控制器将自动前瞻。前瞻的条件是，队列中这些CP指令之间没有JOG、PTP、ARC、WAIT、TRIG等指令。

1.12.2 执行带激光雕刻的 CP 指令

表 1.32 执行带激光雕刻的 CP 指令

原型	int SetCPLECmd (CPCmd *cpCmd, bool isQueued, uint64_t *queuedCmdIndex)
功能	执行带激光雕刻的 CP 指令。此时激光功率将在运动命令下发时起效

参数	CPCmd 定义：
	typedef struct tagCPCmd {
	uint8_t cpMode; //CP 模式。0：相对模式，表示相对距离，即笛卡尔坐标增量。1：绝对模式，表示绝对距离，即笛卡尔坐标系下目标点坐标
	float x; //x,y,z 可以设置为坐标增量，也可设置为目的坐标点
	float y;
	float z;
	union {
	float velocity; //保留
	float power; //激光功率：0~100
	}
	}CPCmd;
	cpCmd：CPCmd 指针
	isQueued：是否将该指令指定为队列命令
	queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义

返回	DobotCommunicate_NoError：指令正常返回 DobotCommunicate_BufferFull：指令队列已满 DobotCommunicate_Timeout：指令无返回，导致超时

1.13 ARC 功能

圆弧模式即示教后存点回放的运动轨迹为圆弧。圆弧轨迹是空间的圆弧，由当前点、圆弧上任一点和圆弧结束点三点共同确定。圆弧总是从起点经过圆弧上任一点再到结束点，如图 1.3所示。

⚠注意

使用圆弧运动模式时，需结合其他运动模式确认圆弧上的三点，且三点不能在同一条直线上。

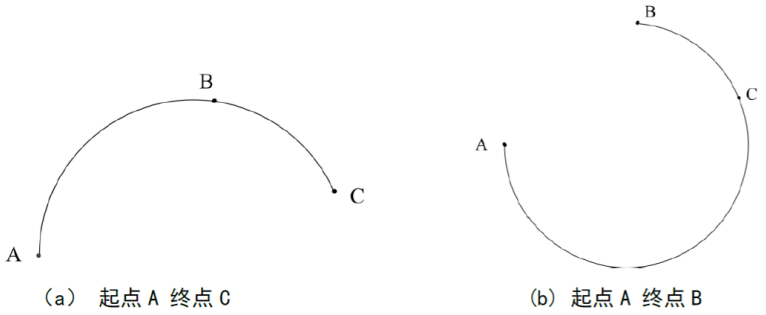


图 1.3 圆弧运动模式

1.13.1 执行 ARC 指令

表 1.33 执行圆弧插补功能接口说明

原型	int SetARCCmd(ARCCmd *arcCmd, bool isQueued, uint64_t *queuedCmdIndex)
功能	执行 ARC 指令。设置 ARC 运动的速度和加速度后，调用该函数可使机械臂按 ARC 模式运动至目标点 该运动模式需结合其他模式一起使用，构成圆弧上三点

参数	ARCCmd 定义：
	typedef struct tagARCCmd {
	struct {
	float x;
	float y;
	float z;
	float r;
	}cirPoint; //圆弧中间点，需设置为笛卡尔坐标
	struct {
	float x;
	float y;
	float z;

	float r;
	}toPoint; //圆弧目标点，需设置为笛卡尔坐标
	}ARCCmd;
	arcCmd：ARCCmd 指针
	isQueued：是否将该指令指定为队列命令
	queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义

返回	DobotCommunicate_NoError：指令正常返回 DobotCommunicate_BufferFull：指令队列已满 DobotCommunicate_Timeout：指令无返回，导致超时

1.13.2 执行 CIRCLE 指令

圆形模式与圆弧模式相似，示教后存点回放的运动轨迹为整圆。使用圆形模式时，也需结合其他运动模式确认圆形上的三点。

表 1.34 执行 CIRCLE 指令

原型	int SetCircleCmd(CircleCmd *circleCmd, bool isQueued, uint64_t *queuedCmdIndex)
功能	执行 CIRCLE 指令。设置整圆运动的速度和加速度后，调用该函数可使机械臂按 CIRCLE 模式运动至目标点 该运动模式需结合其他模式一起使用，构成圆上三点

参数	CircleCmd 定义：
	typedef struct tagCircleCmd{
	struct {
	float x;
	float y;
	float z;
	float r;
	}cirPoint; //圆弧中间点，需设置为笛卡尔坐标
	struct {
	float x;
	float y;
	float z;
	float r;
	}toPoint; //圆弧目标点，需设置为笛卡尔坐标
	uint32_t count; //整圆个数
	}CircleCmd;

	circleCmd：CircleCmd 指针 isQueued：是否将该指令指定为队列命令 queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义
返回	DobotCommunicate_NoError：指令正常返回 DobotCommunicate_BufferFull：指令队列已满 DobotCommunicate_Timeout：指令无返回，导致超时

1.14 WAIT 功能

1.14.1 执行时间等待指令

表 1.35 执行时间等待功能接口说明

原型	int SetWAITCmd(WAITCmd *waitCmd, bool isQueued, uint64_t *queuedCmdIndex)
功能	执行时间等待指令。如果需设置前个指令运行后的暂停时间，可调用此函数 该指令只能作为队列指令，“isQueued”必须设置为“1”。将该命令设置为立即指令可能会导致正在执行的 WAIT 队列指令的暂停时间变化

参数	WAITCmd 定义：
	typedef struct tagWAITCmd {
	uint32_t timeout; //单位：ms
	}WAITCmd;
	waitCmd：时间等待功能变量
	isQueued：是否将该指令指定为队列命令
	queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义

返回	DobotCommunicate_NoError：指令正常返回 DobotCommunicate_BufferFull：指令队列已满 DobotCommunicate_Timeout：指令无返回，导致超时

1.15 触发功能

1.15.1 执行触发指令

表 1.36 执行触发指令

原型	int SetTRIGCmd(TRIGCmd *trigCmd, bool isQueued, uint64_t *queuedCmdIndex)
功能	执行触发功能 该指令只能作为队列指令，“isQueued”必须设置为“1”。将该命令设置

	为立即指令可能会导致正在执行的 TRIG 队列指令的触发条件变化。

参数	TRIGCmd 定义：
	typedef struct tagTRIGCmd {
	uint8_t address; //I/O 地址。若 mode 设置为 0，取值范围：1~24。若 mode 设置为 1，取值范围：1~6
	uint8_t mode; //触发模式。0：电平触发。1：A/D 触发
	uint8_t condition; //触发条件
	电平触发：0，等于。1，不等于
	A/D 触发：0，小于。1，小于等于
	2，大于等于。3，大于
	uint16_t threshold; //触发阈值：电平触发，0 或 1。A/D 触发：0~4095
	}TRIGCmd;
	trigCmd：TRIGCmd 指针
	isQueued：是否将该指令指定为队列命令
	queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义

返回	DobotCommunicate_NoError：指令正常返回 DobotCommunicate_BufferFull：指令队列已满 DobotCommunicate_Timeout：指令无返回，导致超时

1.16 EIO 功能

在 Dobot 控制器中，所有的扩展 I/O 都是统一编址的。根据现有情况，I/O 的功能包括以下内容：

- 高低电平输出功能。
- 读取输入高低电平功能。
- 读取输入模数转换值功能。

I/O 详细说明，请参见《Dobot M1 用户手册》。

1.16.1 设置 I/O 输出电平

表 1.37 设置 I/O 输出电平

原型	int SetIODO(IODO *ioDO, bool isQueued, uint64_t *queuedCmdIndex)
功能	设置 I/O 输出电平

参数	IODO 定义：
	typedef struct tagIODO {
	uint8_t address; //I/O 地址。取值范围：1~22

	uint8_t level; //输出电平。0：低电平。1：高电平
	}IODO;
	ioDO：IODO 指针
	isQueued：是否将该指令指定为队列命令
	queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义

返回	DobotCommunicate_NoError：指令正常返回 DobotCommunicate_BufferFull：指令队列已满 DobotCommunicate_Timeout：指令无返回，导致超时

1.16.2 读取 I/O 输出电平

表 1.38 读取 I/O 输出电平

原型	int GetIODO(IODO *ioDO)
功能	读取 I/O 输出电平

参数	IODO 定义：
	typedef struct tagIODO {
	uint8_t address; //I/O 地址。取值范围：1~22
	uint8_t level; //输出电平。0：低电平。1：高电平
	}IODO;
	ioDO：IODO 指针

返回	DobotCommunicate_NoError：指令正常返回 DobotCommunicate_Timeout：指令无返回，导致超时

1.16.3 读取 I/O 输入电平

表 1.39 读取 I/O 输入电平

原型	int GetIODI(IODI *ioDI)
功能	读取 I/O 输入电平

参数	IODI 定义：
	typedef struct tagIODI {
	uint8_t address; //I/O 地址。取值范围：1~24
	uint8_t level; //输入电平。0：低电平。1：高电平
	}IODI;
	ioDI：IODI 指针

返回	DobotCommunicate_NoError：指令正常返回

	DobotCommunicate_Timeout：指令无返回，导致超时

1.16.4 读取 A/D 输入

表 1.40 读取 A/D 输入

原型	int GetIOADC(IOADC *ioADC)
功能	读取 A/D 输入

参数	IOADC 定义：
	typedef struct tagIOADC {
	uint8_t address; //I/O 地址。取值范围：1~6
	uint16_t value; //A/D 输入值。取值范围：0~4095
	}IOADC;
	ioADC：IOADC 指针

返回	DobotCommunicate_NoError：指令正常返回 DobotCommunicate_Timeout：指令无返回，导致超时

1.17 网络功能

DobotStudio可通过局域网控制Dobot M1。Dobot M1与电脑通过网线连接后，需设置局域网相关参数（IP地址、子网掩码、默认网关等），使Dobot M1接入局域网。接入成功后Dobot M1无需通过USB即可连接M1Studio。

1.17.1 配置 LAN

表 1.41 配置 LAN

原型	int SetFirmwareLanConfig(LanConfig * config)
描述	设置LAN配置。

参数	LanConfig定义：
	typedef struct tagLanConfig {
	uint8_t status; //保留
	bool dhcp; //是否开启DHCP。0：关闭。1：开启
	uint8_t addr[16]; //IP地址
	uint8_t mask[16]; //子网掩码
	uint8_t gateway[16]; //网关
	uint8_t dns[16]; //DNS
	}LanConfig;
	config：LanConfig指针

返回	DobotCommunicate_NoError：指令正常返回

	DobotCommunicate_Timeout：指令无返回，导致超时

1.17.2 获取 LAN 配置

表 1.42 获取 LAN 配置

原型	int GetFirmwareLanConfig (LanConfig * config)
描述	获取LAN配置

参数	typedef struct tagLanConfig {
	uint8_t status; //连接状态
	0：断开连接
	1：连接
	2：正在搜索
	3：正在连接
	4：不使能
	5：错误
	bool dhcp; //是否开启DHCP。0：关闭。1：开启
	uint8_t addr[16]; //IP地址
	uint8_t mask[16]; //子网掩码
	uint8_t gateway[16]; //网关
	uint8_t dns[16]; //DNS
	}LanConfig;
	config：LanConfig指针

返回	DobotCommunicate_NoError:指令正常返回 DobotCommunicate_Timeout:指令无返回，导致超时

1.18 其他功能

1.18.1 事件循环功能

在某些语言中，当调用 API 接口后，如果没有事件循环，应用程序将直接退出，导致指令没有下发至 Dobot 控制器。为避免这种情况发生，我们提供了事件循环接口，在应用程序退出前调用（目前已知需要做此处理的语言有 Python）。

表 1.43 事件循环功能接口说明

原型	void DobotExec(void)
功能	事件循环功能

参数	无
返回	无

